MSRS 시뮬레이션 환경에서 가상 로봇의 네트웍제어

Network Control for Virtual Robot in MSRS Simulation Environment

신 동 관[1]·이 성 훈[2]·이 수 영[3]·최 병 욱†

Dong-Gwan Shin[1]·Sung-Hun Lee[2]·Soo-Yeong Yi[3]·Byoung-Wook Choi†

Abstract Robot system development consists of several sub-tasks such as layout design, motion planing, and sensor programming etc. In general, on-line programming and debugging for such tasks demands burdensome time and labor costs, which motivates an off-line graphic simulation system. MSRS(Microsoft Robotics Studio) released in recent years is an appropriate tool for the graphic simulation system since it supports CCR(Concurrency and Coordination Runtime), DSS(Decentralized System Services), and dynamics simulation based on PhysX and graphic animation as well. In this paper, we developed an MSRS based network simulation system for quadruped walking robots, which controls virtual 3D graphic robots existing in remote side through internet.

Keywords : 3D simulation, Microsoft Robotics Studio, virtual robot, network control

1. 서 론

전기·전자 및 정보 분야에서의 기술혁신에 의해 일부 산업현장 및 극한환경에서 사용되었던 로봇을 인간 생활에 이용하여 삶을 더욱더 윤택하게 하고자 하는 연구들이 최근 많은 연구실 및 산업체에서 행해지고 있다[1][2][3]. 이러한 로봇을 개발하는데 있어서 로봇의 움직임을 결정하고 배치할 때 개발자가 로봇의 움직임을 직접 확인하면서 위치를 수정하고 프로그램을 변경하는 작업은 많은 시간과 노력이 소요되고 그에 따른 비용이 추가된다.

이러한 문제를 해결하기 위한 방법에는 여러 가지가 있겠지만 그 중 3차원 공간에서의 모의 시험하는 방법을 생각할 수 있다. 3차원 공간에서의 시뮬레이션은 기존에 가지고 있던 로봇의 정보 및 새로 추가된 여러 정보기술들을 미리 시스템에 도입하여 가상적으로 로봇을 작동해 볼 수 있는 컴퓨터 모델이다. 이것은 로봇을 가상환경으로 그대로 옮겨 놓은 것이므로 실제와 동일하다고 간주할 수 있다. 이를 이용하면 수치적인 방법으로는 파악하기 힘든 구조적인 충돌, 간섭 또한 쉽게 찾아낼 수 있으며 로봇 개발과정에서 빈번하게 발생하고 있는 설계변경 및 프로그램 수정에 추가되는 비용과 시간의 낭비를 최소화할 수 있다.

다시 말하자면 3차원 공간의 시뮬레이션을 통한 개발은 개발자의 의사결정의 질을 향상시키기 위한 통합 모의환경으로 이는 실제 환경에 근접한 컴퓨터 모델을 구축하고 이를 이용하여 실제 로봇을 수정하고 제어함으로써 로봇의 개발 기간 및 비용을 줄이기 위한 방법이라고 할 수 있다.

본 논문에서는 네트워크로 연결된 두 대의 컴퓨터에 시뮬레이션 환경의 가상 로봇을 구현하여 네트워크 게임처럼 두 대의 컴퓨터가 서로 통신하여 각자의 로봇을 제어하도록 하였다. 본 논문에서 사용한 시뮬레이션 툴은 MSRS(Microsoft Robotics Studio)에서 제공하는 시뮬레이션 툴을 사용하였다. 이 시뮬레이션 툴은 AGEIA 사의 PhysX SDK를 통해 사물에 물리적인 특성을 부여할 수 있어 로봇의 충돌 및 자세제어에 대한 동작 시뮬레이션을 수행 할 수 있다.

본 논문의 구성은 다음과 같이 구성하였다. 2장에서는 우리가 사용한 MSRS에 대하여 알아본다. 3장에서는 우선 시뮬레이션 환경에 로봇을 어떻게 추가하는지 알

[1] 서울산업대학교 전기공학과 석사과정(E-mail : shingun@snut.ac.kr)
[2] 서울산업대학교 전기공학과 석사과정(E-mail : dashe@snut.ac.kr)
[3] 서울산업대학교 전기공학과 부교수(E-mail : suylee@snut.ac.kr)
† 교신저자 : 서울산업대학교 전기공학과 부교수
(E-mail : bwchoi@snut.ac.kr)

아보고, 보안 설정 및 제어프로그램 수정을 통한 네트웍 제어 방법에 대해 알아본다. 마지막으로 4장에서는 본 논문의 결론을 정리한다.

2. Microsoft Robotics Studio

오늘날의 로봇 산업이 1970년대의 PC산업과 비슷한 양상을 보인다는 분석과 함께 앞으로의 로봇산업의 발전 가능성을 본 마이크로소프트사는 지난 2006년 12월에 MSRS 1.0을 시작으로 현재 MSRS 1.5버전까지 공개했다[4].

현재 로봇을 개발하는 각각의 회사는 서로 다른 운영체제와 하드웨어를 가지고 로봇 개발을 수행한다. 물론 프로그램 언어 또한 다르다. 이러한 다른 로봇 시스템 사이의 어플리케이션 코드의 재사용은 거의 불가능 하다. 마이크로소프트에서는 이러한 문제를 닷넷 프레임 워크를 사용하여 해결하고자 하였다.

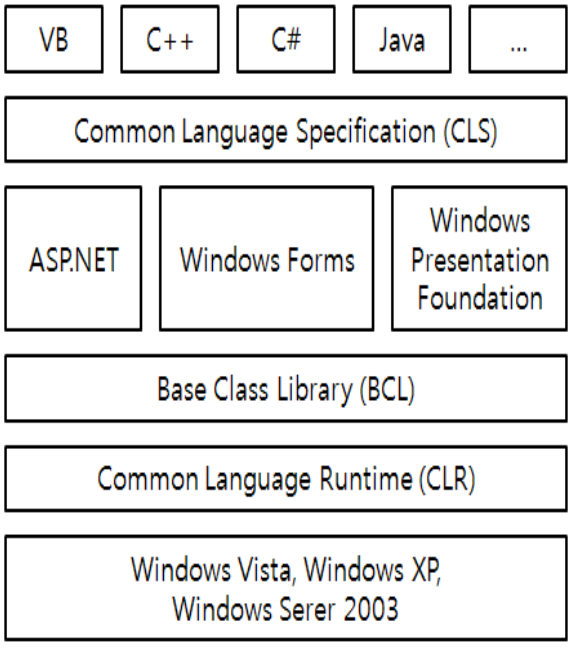

그림 1. 닷넷 프레임 워크 구조

MSRS는 학생, 상업적인 개발자등 다양한 사람들이 다양한 하드웨어를 통해 쉽게 로봇 어플리케이션을 만들 수 있는 윈도우 기반의 환경이다. MSRS 로봇 어플리케이션의 전체적인 목적은 조정(orchestrate)에서 다양한 센서로부터 입력받은 정보를 가지고 모터를 구동하는 것으로 그림 2에서는 이에 해당하는 간단한 예를 보이고 있다.

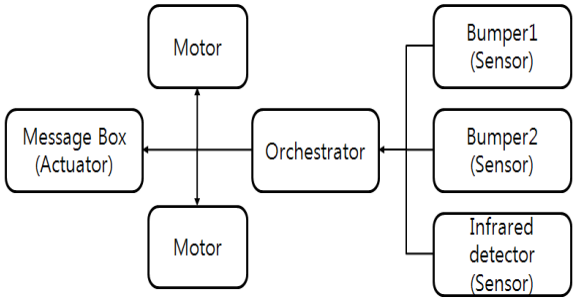

그림 2. MSRS Runtime 예제

또한 CCR(Concurrency and Coordination Runtime) 을 사용함으로써 쓰레드(thread), 록(lock), 세마포어(semaphore) 등의 사용 없이 메시지를 동시에 처리할 수 있다.

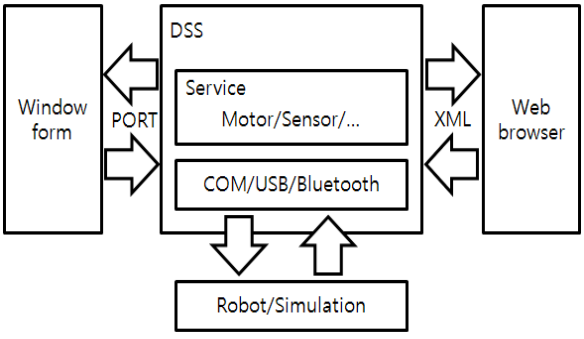

그림 3. MSRS를 이용한 로봇 시스템 아키텍처

그리고 DSS(Decentralized System Services)를 적용함으로써 서비스 호스팅 환경을 제공하고 디버깅, 기록, 감시, 보안 등과 같은 용이한 작업들을 제공한다.

3. 시뮬레이션 환경 구현

본 논문에서는 강아지 형태의 다관절 로봇을 3D 모델링하여 물리력이 작용하는 시뮬레이션 환경에 적용하였다. 가상 로봇을 구현하기 위해 로봇 객체를 생성하여 로봇의 무게, 표면(Texture) 및 관절 연결에 관한 정보를 작성하였다.

또한, 가상 로봇을 네트웍으로 제어할 수 있도록 하기위해 SimpleDashboard 프로젝트를 수정하여 원격 컴퓨터와 호스트 컴퓨터의 가상 로봇을 동시에 제어 할 수 있도록 하였다.

이와 같은 작업을 수행하기 위해 우리는 MSRS 웹사이트[5]에서 MSRS 1.5를 다운로드하여 설치하였고, 컴파일러는 마이크로소프트사에서 권장하고 있는 Visual Studio 2005 Professional Edition을 사용하였다.

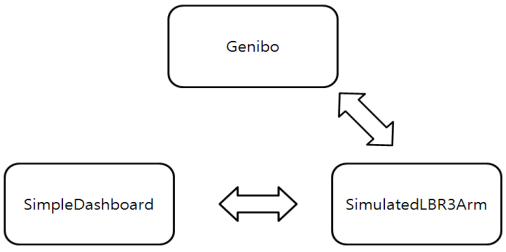

그림 4. 서비스 구성도

3.1 3D 모델링을 통한 가상 로봇 Entity 구현

MSRS에서 제공하는 시뮬레이션에 가상 로봇을 구현하기 위해서 우리는 [그림 5]와 같은 작업을 수행 하였다.

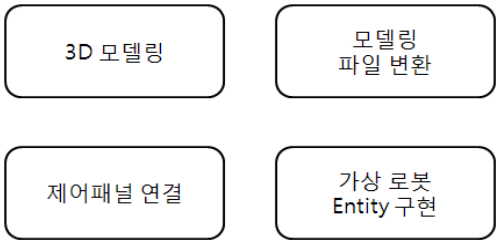

그림 5. 가상 로봇 구현

로봇의 3D 모델링 작업은 PRO-E를 사용하였고 PRO-E를 사용하여 만든 모델링 파일을 MSRS에서 사용하기 위해서는 파일 변환 과정이 필요하다. 이때 모델링 파일을 변환하기 위해 사용한 도구는 MSDN에서 소개된 Blender를 사용하였는데 이것은 PRO-E의 모델링 파일을 객체(obj) 파일로 변환하여 MSRS에서 로봇의 표면에 사용된다.

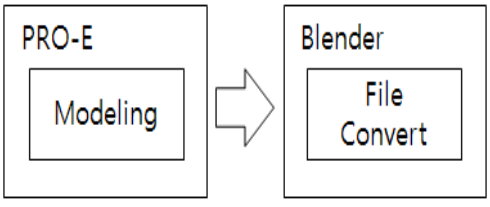

그림 6. 모델링 파일 변환

가상 로봇을 구현하기 위해서는 우선 로봇 Entity가 존재할 프로젝트를 생성해야 한다. MSRS에서는 이러한 작업을 수월하게 하기 위해 dssnewservice 라는 명령어를 제공한다. 이 명령어는 다음과 같이 사용하며 SimulatonSNUT라는 namespace와 Genibo라는 서비스를 생성하게 된다.

```
..>cd Samples
..₩Samples>dssnewservice /namespace:SimulationSNUT /service:Genibo
```

생성한 프로젝트에 시뮬레이션 환경을 적용하기 위해 필요한 라이브러리를 추가한다. 또한 추가한 라이브러리를 사용하기 위해 using 키워드를 사용하여 소스의 상단에 기입하여 준다.

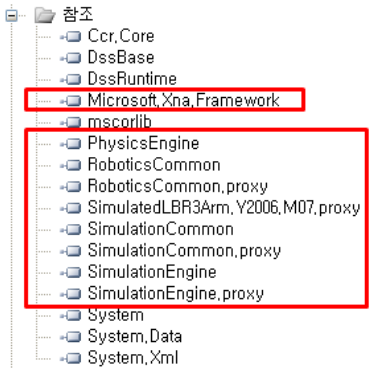

그림 7. 라이브러리 추가

로봇의 객체를 구현하기 위해 프로젝트에 객체 클래스를 추가 생성한 후 앞서 말한 것과 같이 시뮬레이션에 필요한 라이브러리들을 사용하기 위해 소스의 상단에 using 키워드를 사용하여 기입하여 준다.

우리가 구현하려고 하는 가상 로봇은 다관절 로봇으로서 이는 MSRS의 시뮬레이션 튜토리얼 4에 있는 ArticulatedArms과 그 기본 구조가 비슷하여 그에 해당하는 객체파일을 가져다가 수정하였다. 이 객체 정보는 MSRS의 하위폴더인 samples\simulation\Entityes에 존재하며 이 파일에는 ArticulatedArms 객체 이외에도 센서, 모터, 기타 로봇 플랫폼 등의 정보도 수록되어있다.

가상 로봇을 구현하기 위해 생성한 로봇 객체파일에서 우리가 해주어야 할 일을 그림 8에서 보이고 있다. 우선 베이스에 대한 물리 정보를 부여하고 이를 바탕으로 나머지 관절에 대한 물리 정보를 정의한 후 Blender로 변환한 3D 모델링 파일을 사용하기 위한 변수를 선언한다. 그 다음 관절의 연결 정보를 수록하는 연결부의 변수를 선언하는데 우리는 강아지 형태의 로봇을 구현하기 위해 5개의 변수를 선언하였다. 메인소스로부터 로봇의 초기 생성 위치를 전달 받는 이 함수는 원래 표면 정보에 관한 전달은 없었다. 기본 객체파일에서 표면은 연결부를 생성해 주는 부분에서 바로 적용하였는데 이렇게 처리될 경우 같은 로봇을 여러 개 생성할 때 각각의 로봇의 표면을 다르게 적용할 수 없게 된다. 때문

에 우리는 변수를 선언하고 이 함수에 표면 전달인자를 추가하였다. 그 다음 앞에서 정의한 각 관절의 물리정보와 표면 정보를 이용하여 각각의 다리에 대한 링크를 구성하고 제어 프로그램에서 각 관절을 제어할 수 있도록 이름 부여 및 각 연결부를 초기화 해주는 초기화 함수를 구현한다. 이 후에 있는 속도 및 자세 등을 처리해 주는 함수 및 구성한 연결부를 배열하는 함수는 기존의 객체파일에 있는 함수 그대로 사용하여도 된다.

```
Public class GeniboEntity : VisualEntity
{
    const float baseMass, baseThickness, baseLength ;
    ...
    string Texture1, Texture2, ... ;

    List(ArmLinkEntity> _links =
            new List<ArmLinkEntity>();

    public GeniboEntity(Vector3 postion,
            string texture1, string texture2, ...);
    {
        Texture1 = texture1;
        ...
    }

    void CreateLink(Joint joint) {...}

    public override void Initialize
            (xnagrfx.GraphicsDeve device,
            PhysicsEngine physicsEngine) {...}

    public void SetJointTargetOrientation() {...}

    public void SetJointTargetPose() {...}

    ...

    public override void Dispose() {...};
}
```

그림 8. Visual Robot Entity Code

```
Public class GeniboService : DsspServiceBase
{
    [Partner("Engine",
      Contract = engeneproxy.Contract.Identifier,
      CreationProxy = PartnerCreationPolicy.UseExistingOrCreate)]

    arm.ArticulatedArmOperations _armServicePort;

    private override void Start()
    {
        base.Start();
        SetupCamera();
        PopulateWorld();
    }

    SetupCamera(){...};

    PopulateWorld()
    {
        AddSky();
        AddGround();
        SpawnIterator(AddGenibo);
    }

    Ienumerator<Itask> AddGenibo()
    {
        ...
    }
}
```

그림 9. Main Source Code

그림 9는 앞에서 생성한 프로젝트의 메인 소스로 시뮬레이션에 존재하는 각종 사물들을 선언해 주며 로봇의 ID를 제어 프로그램으로 전달하는 과정을 보이고 있다.

우선 시뮬레이션 엔진을 사용하기 위한 파트너를 선언하였고, 제어 프로그램과의 데이터 통신을 위해 _armServcePort를 선언하였다. Start() 함수는 이후에 정의한 AddSky(), AddGround() 등과 같은 시뮬레이션 환경 함수와 가상로봇 생성 함수를 구동하여 프로그램 운용 시 시뮬레이션 상에 나타낼 수 있도록 해준다. 이 시뮬레이션 환경 함수는 MSRS의 튜토리얼에 있는 함수를 가져다가 사용하였다. 가상 로봇 생성함수 역시 MSRS의 시뮬레이션 튜토리얼4에 있는 내용을 기초로 하여 작성하였다. 이 함수에서는 가상 로봇 객체 파일로 넘겨줄 로봇의 초기 위치 및 표면에 대한 값을 정의한다. 그 다음 생성할 로봇의 이름을 부여한 후 시뮬레이션 서비스를 생성하고 앞에서 선언한 _armServicePort를 통해 서비스 생성 결과를 전달한다.

이렇게 전달된 정보는 SimulatedLBR3Arm 프로젝트로 전달된다. 이 프로젝트는 가상 로봇 서비스를 제어프로그램인 SimpleDashboard와 연결해 주는 역할을 한다. 여기에서 우리가 수정해 줄 것은 거의 없다. 우리의 프로젝트가 기존의 ArticulatedArm 프로젝트를 기반으로 작성한 것이라 기존의 함수를 그대로 사용하여도 무방한 것이다. 우리가 수정해 줄 것은 단지 생성한 로봇 객체 라이브러리를 참조해주고 기존에 사용하던 Kuka 로봇에 대한 객체 이름을 우리의 Genibo로 교체해 주기만 하면 된다.

이러한 작업들을 수행함으로써 물리력이 작용하는 시뮬레이션 환경에서 동작하는 가상 로봇을 구현할 수 있었다. 그림 10은 시뮬레이션에서 구동하고 있는 가상 로봇을 보이고 있으며 이것은 제어프로그램인 SimpleDashboard를 통하여 각 관절을 제어할 수 있도록 하였다.

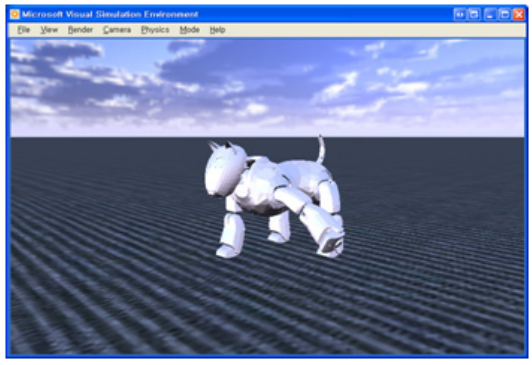

그림 10. 시뮬레이션으로 구현한 가상 로봇

3.2 네트웍 컴퓨터의 가상 로봇 제어

네트워크로 연결된 두 대의 컴퓨터에서 서로 다른 가상 로봇을 제어하기 위해서는 Security Setting 과정이 필요하다. 보안 설정 방법이 두 가지가 있는데 그 중 하나는 MSRS\store\SecuritySettings.xml 파일에서 AuthenticationRequired 항목을 “false“로 변경하는 것이다. 또 다른 하나는 시뮬레이션 구동 후 웹 브라우저에 http://localhost:50000/ 을 기입하면 현재 구동하고 있는 시뮬레이션에 관한 상태 정보가 나타난다. 여기서 좌측에 있는 메뉴에 보안 관리자 항목이 있는데 이것을 선택하면 보안 설정을 할 수 있다. 여기서 ”Authentication” 을 ”Disabled”로 설정하면 된다.

보안 설정 과정을 수행한 후 네트워크로 연결된 다른 컴퓨터의 가상 로봇을 제어하기 위한 접속을 잘 이루어진다. 하지만 가상 로봇의 제어는 이루어지지 않는다. 이러한 문제를 해결하기 위해서는 제어판 - 윈도우 방화벽 - 예외 - 파일 및 프린터 공유에 체크를 해주어야 한다. 이와 같은 작업을 수행하면 가상 로봇의 네트웍제어가 가능하게 된다.

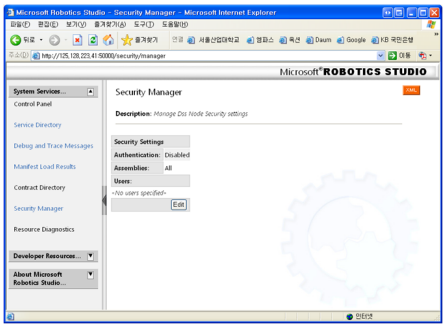

그림 11. Security Setting

하지만 이러한 네트웍제어는 컴퓨터 A에서 컴퓨터 B로 원격 접속하여 제어할 경우 컴퓨터 A에서 보여지는 시뮬레이션의 가상 로봇은 작동하지 않는 문제점이 발생한다.

이러한 문제를 해결하기 위해 우리는 그림 12.와 그림 13과 같은 생각을 해보았다. 그림 12는 호스트 컴퓨터에서 네트웍 컴퓨터의 가상 로봇으로 접속하는 방식으로 이는 기존의 네트웍 접속 방식과 유사하다. 하지만 기존의 방식과 다른 점은 원격의 가상 로봇을 제어하면서 동시에 호스트의 가상 로봇을 같이 제어하자는 것이다. 이것은 제어 프로그램에서 원격의 가상 로봇에게 명령을 보낼 때 호스트의 가상 로봇에도 같이 명령을 보내 동시에 제어하고자 하는 것인데 여기에 약간의 문제가 있다. 그것은 제어하고자하는 호스트의 가상 로봇의 ID를 읽어올 수 가 없다는 것이다. 때문에 우리는 두 번째 계획안을 수행하였다.

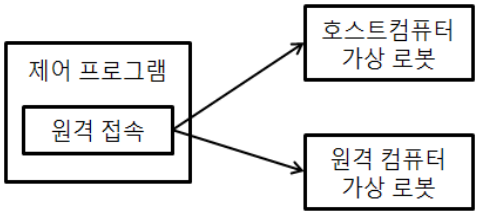

그림 12. 양방향 제어 계획안(1)

두 번째 계획안이 첫 번째 계획안과 다른 점은 제어프로그램에서 가상 로봇에 접근할 때 호스트와 원격 양쪽으로 접속한다는 것이다. 이러한 방법은 앞에서 논의한 두 대의 컴퓨터의 가상 로봇을 동시에 제어하는 문제와 계획 1안에서 야기된 ID 문제 역시 해결할 수 있다.

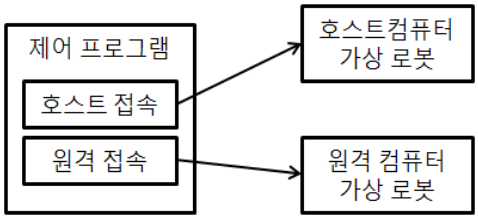

그림 13. 양방향 제어 계획안(2)

두 번째 계획안을 수행하기 위해 우리는 제어 프로그램을 수정해야한다. 여기서 말하는 제어 프로그램은 MSRS에서 제공하는 기본 프로그램으로 모든 시뮬레이션 튜토리얼에서 사용하고 있는 SimpleDashboard이다.

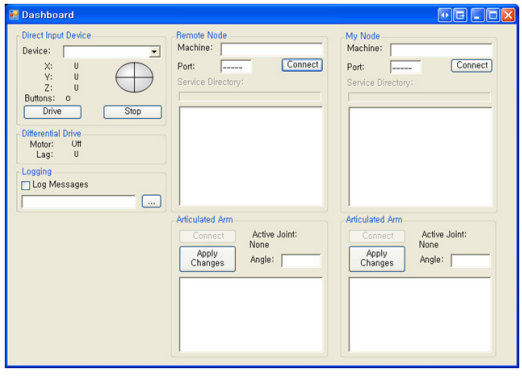

그림 14. SimpleDashboard Layout

제어 프로그램에서 기존의 접속 및 제어 부분을 복사하여 원격 컴퓨터와 호스트 컴퓨터로의 접속을 가능하

도록 하였다. 윈도우 폼에서 이벤트가 발생하였을 경우 해당 이벤트는 이벤트 통로를 통하여 simpledashboard의 메인 소스의 처리부로 전달된다. 전달된 정보는 _articulatedArmPort를 통하여 로봇 객체로 전달되거나 DriveControl를 통하여 다시 윈도우 폼으로 전달되어 그에 해당하는 처리를 수행하게 된다. 그림 15는 이러한 simpledashboard의 데이터 흐름을 보이고 있다.

제어 주 프로그램에서 수정할 내용은 그림 16에서 보이고 있다. 우선 원격 컴퓨터의 제어와 더불어 호스트 컴퓨터의 제어를 위한 DriveControl 변수를 추가하였고, Start() 함수에서는 이벤트 통로를 통하여 받은 윈도우 폼에서 발생한 이벤트를 처리부로 연결하는 부분에 호스트에 관련된 부분을 추가하여 주었다. 끝으로 해당 이벤트에 대한 처리부를 구현해 줌으로써 가상 로봇의 양방향 제어를 수행할 수 있었다.

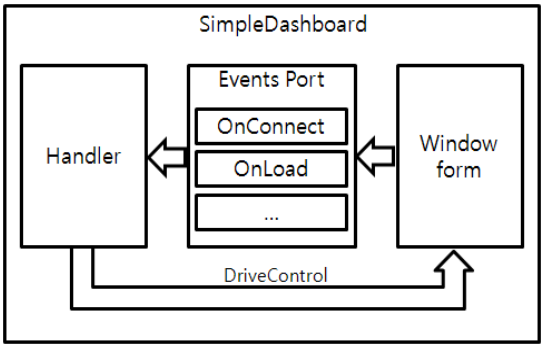

그림 15. SimpleDashboard Data Flow

```
Public class SimpleDashboardService : DsspServiceBase
{
    DriveControl _driveControl;

    protected override void Start()
    {
        Activate(Arbiter.Interleave(
            new TeardownReceiverGroup{...},
            new ExclusiveReceiverGroup{...},
            new ConcurrentReceiverGroup
                {
                    Arbiter.ReceiveWithIterator<OnConnect>
                        (true, _eventsPort, OnConnectHandler),
                    ...
                }
        ));
    }

    Ienumerator<Itask> OnConnectHandler(Onconnect onConnect)
    {
        ...
    }
}
```

그림 16. SimpleDashboard Source Code

[그림 17]은 네트워크에 연결된 두 대의 컴퓨터가 서로 네트웍 제어를 통해 가상 로봇을 제어하는 모습이다.

그림 17. 네트워크로 연결된 컴퓨터에서 가상 로봇의 네트웍 양방향 제어

4. 결 론

사회에서 로봇에 대한 관심이 높아짐에 따라 산업체에서는 로봇의 오작동으로 인한 인적, 물적 손실을 줄이고 제품 개발 주기의 단축 등을 위해 많은 시간과 노력 및 비용을 투자하고 있다.

하지만 현재 로봇 개발에 있어서 일반적으로 하드웨어 로봇은 고가의 부품들로 구성되며, 하드웨어 로봇 개발과 소프트웨어 개발이 병행되지 못하고 순차적으로 진행됨으로써, 비용과 개발 기간이 많이 소요되고 있다. 또한 재사용되지 못하는 코드들로 인해 매번 동일한 작업을 다시 반복하게 됨으로써 비용과 기간의 증가를 유발한다.

MSRS는 2장에서 언급한 DSS와 CCR을 통해 재사용성과 코딩의 용이성을 제공하며 시뮬레이션 환경을 통해 하드웨어 로봇과 소프트웨어 개발의 병행이 가능하도록 하고, 소프트웨어를 테스트 할 수 있는 시뮬레이션 환경을 제공함으로써 비용과 기간을 대폭 감소시키는 효과를 얻을 수 있다.

본 논문에서는 이와 같은 MSRS의 특징을 적용하여 물리적 특성을 부여할 수 있는 3차원 시뮬레이션 공간에 가상 로봇을 구현하였다. 이러한 시뮬레이션에서 로봇의 모션 제어를 수행해 봄으로써 로봇의 기구학적 오류를 초기에 발견해 낼 수 있고, 로봇의 각 부품에 대한 시뮬레이션 코드를 재사용함으로써 제품의 개발 비용 및 기간을 단축할 수 있으리라고 본다. 또한 가상 로봇의 양방향 제어를 수행해 봄으로써 카메라가 없는 환경에서 네트워크로 연결된 네트웍 로봇의 제어 가능성을 보았다.

하지만 최적화 되지 못한 서비스의 분산은 전체 프로그램을 이해하는데 어려움을 주고 있다. 또한 가상 로봇의 모션제어 역시 미흡한 상태이다. 따라서 추후에는 로

봇 객체와 SimulatedLBL3Arm을 통합하여 서비스의 무의미한 분산을 억제하고 제어 프로그램을 최적화할 것이다. 또한 좀 더 사실적인 모션제어를 통해 실제 로봇과 가상 로봇을 동시에 제어했을 경우 시뮬레이션 뷰어를 통해 충분히 모니터링 할 수 있도록 하고자 한다.

참 고 문 헌

[1] 남상엽 외, "ED-7271을 이용한 지능형로봇 구조 및 응용", 2006.

[2] http://www.urcdc.or.kr/

[3] 전략기술경영연구원, "유비쿼터스 - 지능형 로봇", 2004

[4] http://irobas.co.kr/2006/kor/about/info06.asp

[5] http://msdn2.microsoft.com/ko-kr/robotics/

[6] http://forums.microsoft.com/default.aspx?ForumGroupID=383&SiteID=1

[7] http://channel9vip.orcsweb.com/wiki/default.aspx/Channel9.MSRoboticsStudio

[8] http://channel9.msdn.com/wiki/default.aspx/Channel9.SimulationImportTutorials

[9] Thomas Stumpfegger, Adnreas Tremmel, Christian Tarragona, and Michael Haag, "A Virtual Robot Control Using a Service-Based Architecture and a Physics-Based Simulation Environment"

[10] http://www.blender.org/

최 병 욱

1986 한국항공대학교 전자공학과(학사)

1988 한국과학기술원 전기 및 전자공학과(공학석사)

1992 한국과학기술원 전기 및 전자공학과(공학박사)

1988~2000 LG 산전주식회사, 엘리베이터 연구실장 및 임베디드 시스템 연구팀장

2000~2005 선문대학교, 제어계측공학과 부교수

2005~현재 서울산업대학교, 전기공학과 부교수

관심분야 : 임베디드 시스템, 실시간 제어 시스템, 지능형 로봇

이 수 영

1988 연세대학교 공과대학 전자공학과(학사)

1990 한국과학기술원 전기 및 전자공학과(공학석사)

1994 한국과학기술원 전기 및 전자공학과(공학박사)

1995~1999 한국과학기술연구원 휴먼로봇연구센터 선임연구원

1999~2007 전북대학교 전자정보공학부 부교수

2007~현재 서울산업대학교, 전기공학과 부교수

관심분야 : Walking robot system, Gait design and motion control, Robot vision

신 동 관

2006 선문대학교 기계 및 제어공학부(학사)

2006~현재 서울산업대학교 전기공학과 석사과정

관심분야 : 임베디드 시스템, 유비쿼터스 컴퓨팅, 지능형 서비스 로봇

이 성 훈

2006 선문대학교 기계 및 제어공학부 (학사)

2006~현재 서울산업대학교 전기공학과 석사과정

관심분야 : 지능형 서비스 로봇, Robot Vision